



ITRI613 Databases I

Chapter 3 – The Relational Model



Learning outcomes

After engaging with the materials and activities in this study unit you should be able to:


- describe the characteristics of the relation model;
- apply the necessary procedures and instructions to force the integrity restrictions decided upon;
- write down the necessary instructions so that a database is created with the respective tables according to the design of the ER diagram and other specified business rules.

Why Study the Relational Model?

- Most widely used model.
 - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- “Legacy systems” in older models
 - e.g., IBM’s IMS
- Recent competitor: object-oriented model
 - ObjectStore, Versant, Ontos, db4o, Smalltalk
 - A synthesis emerging: *object-relational model*
 - Informix Universal Server, UniSQL, O2, Oracle, DB2

Relational Database: Definitions

- *Relational database*: a set of *relations*
- *Relation*: made up of 2 parts:
 - *Instance* : a *table*, with rows and columns.
#Rows = *cardinality*, #fields = *degree / arity*.
 - *Schema* : specifies name of relation, plus name and type of each column.
 - e.g. Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).
- Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).



Never store
age, rather
DOB - 😊LR

Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- ❖ Cardinality = 3, degree = 5, all rows distinct
- ❖ Do all columns in a relation instance have to be distinct?

Relational Query Languages

- A major strength of the relational model: *supports simple, powerful querying of data.*
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
 - The key: *precise semantics* for relational queries.
 - Allows the *optimizer to extensively re-order operations*, and still ensure that the answer does not change.

The SQL Query Language

- Developed by IBM (system R) in the 1970s
- Need for a standard since it is used by many vendors
- Standards:
 - SQL-86
 - SQL-89 (minor revision)
 - SQL-92 (major revision)
 - SQL-99 (major extensions, current standard)

The SQL Query Language

- To find all 18 year old students, we can write:

```
SELECT *  
FROM Students S  
WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```


Querying Multiple Relations

- What does the following query compute?

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid AND E.grade="A"
```

Given the following instances of Enrolled and Students:

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

we get:

S.name	E.cid
Smith	Topology112

Creating Relations in SQL

Rather use
varchar for
name - 😊LR

Not correct
datatypes
for SQL -
😊LR

- Creates the Students relation. Observe that the type (**domain**) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students  
(sid: CHAR(20),  
name: CHAR(20),  
login: CHAR(10),  
age: INTEGER,  
gpa: REAL)
```

- As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Enrolled  
(sid: CHAR(20),  
cid: CHAR(20),  
grade: CHAR(2))
```

Destroying and Altering Relations

DROP TABLE Students

- Destroys the relation Students. The schema information *and* the tuples are deleted.

ALTER TABLE Students

ADD COLUMN firstYear: integer

- ❖ The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- ❖ Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

** Powerful variants of these commands are available; more later!*

Integrity Constraints (ICs)

- **IC:** condition that must be true for *any* instance of the database; e.g., domain constraints.
 - ICs are specified when schema is defined.
 - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
 - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
 - Avoids data entry errors, too!

Primary Key Constraints

- A set of fields is a key for a relation if :
 1. No two distinct tuples can have same values in all key fields, and
 2. This is not true for any subset of the key.
 - Part 2 false? A *superkey*.
 - If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.
- E.g., *sid* is a key for Students. (What about *name*?) The set {*sid*, *gpa*} is a superkey.

Primary and Candidate Keys in SQL

- Possibly many candidate keys (specified using **UNIQUE**), one of which is chosen as the *primary key*.
- ❖ “For a given student and course, there is a single grade.” **vs.** “Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade.”
- ❖ Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled  
(sid CHAR(20)  
  cid CHAR(20),  
  grade CHAR(2),  
  PRIMARY KEY (sid,cid) )
```

```
CREATE TABLE Enrolled  
(sid CHAR(20)  
  cid CHAR(20),  
  grade CHAR(2),  
  PRIMARY KEY (sid),  
  UNIQUE (cid, grade) )
```

Foreign Keys, Referential Integrity

- Foreign key : Set of fields in one relation that is used to `refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer'.
- E.g. *sid* is a foreign key referring to **Students**:
 - Enrolled(*sid*: string, *cid*: string, *grade*: string)
 - If all foreign key constraints are enforced, referential integrity is achieved, i.e., no dangling references.
 - Can you name a data model w/o referential integrity?
 - Links in HTML!

Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

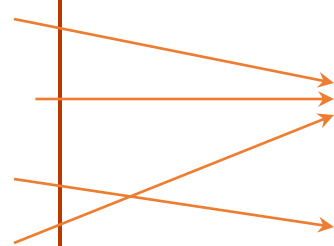
```
CREATE TABLE Enrolled  
  (sid CHAR(20), cid CHAR(20), grade CHAR(2),  
   PRIMARY KEY (sid,cid),  
   FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8



Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
 - Also delete all Enrolled tuples that refer to it.
 - Disallow deletion of a Students tuple that is referred to.
 - Set *sid* in Enrolled tuples that refer to it to a *default sid*.
 - (In SQL, also: Set *sid* in Enrolled tuples that refer to it to a special value *null*, denoting 'unknown' or 'inapplicable'.)
- Similar if primary key of Students tuple is updated.

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
 - Default is **NO ACTION** (*delete/update is rejected*)
 - **CASCADE** (also delete all tuples that refer to deleted tuple)
 - **SET NULL / SET DEFAULT** (sets foreign key value of referencing tuple)

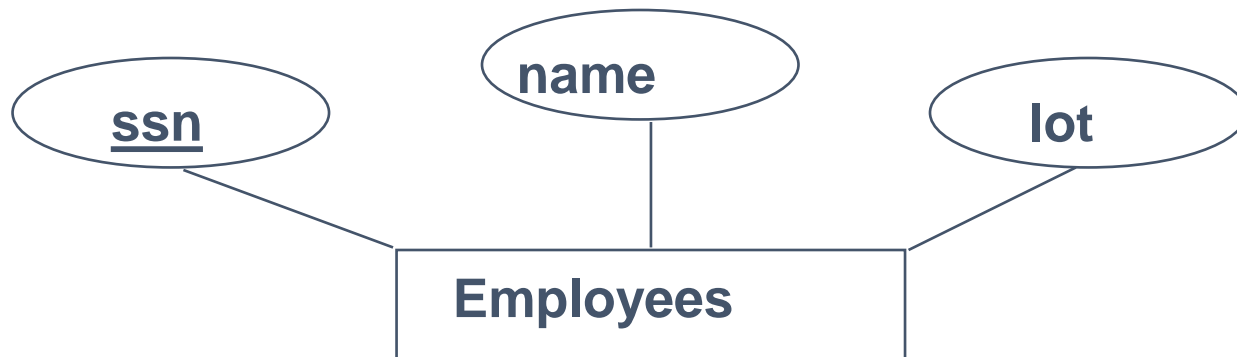
```
CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid)
REFERENCES Students
ON DELETE CASCADE
ON UPDATE SET DEFAULT )
```

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can **NEVER** infer that an IC is true by looking at an instance.
 - An IC is a statement about *all possible* instances!
 - From example, we know *name* is not a key, but the assertion that *sid* is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.

Logical DB Design: ER to Relational

- Entity sets to tables:



```
CREATE TABLE Employees  
  (ssn CHAR(11),  
   name CHAR(20),  
   lot INTEGER,  
   PRIMARY KEY (ssn))
```

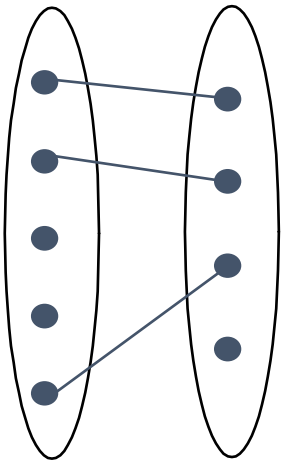
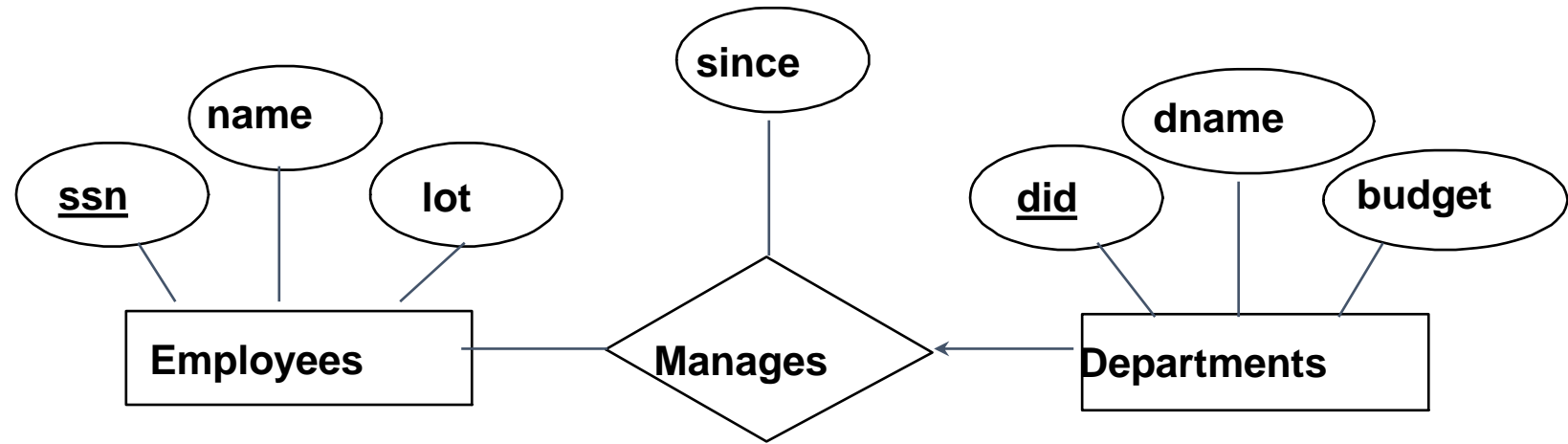
Relationship Sets to Tables

- In translating a relationship set to a relation, attributes of the relation must include:
 - Keys for each participating entity set (as foreign keys).
 - This set of attributes forms a *superkey* for the relation.
 - All descriptive attributes.

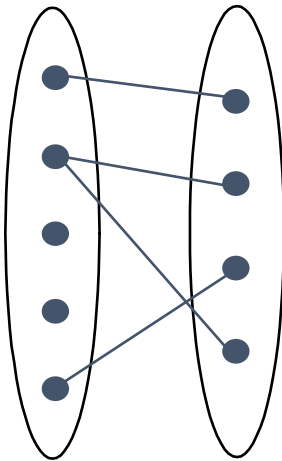
```
CREATE TABLE Works_In(  
    ssn CHAR(11),  
    did INTEGER,  
    since DATE,  
    PRIMARY KEY (ssn, did),  
    FOREIGN KEY (ssn)  
        REFERENCES Employees,  
    FOREIGN KEY (did)  
        REFERENCES Departments)
```

Review: Key Constraints

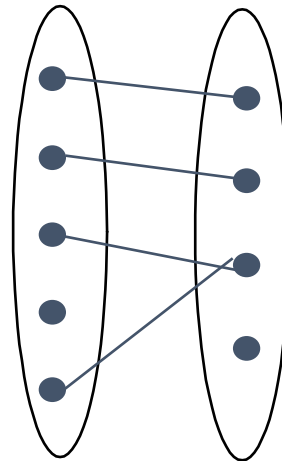
- Each dept has at most one manager, according to the *key constraint* on Manages.



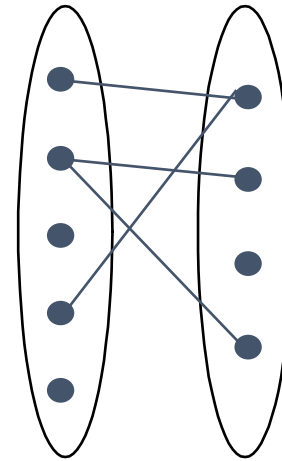
1-to-1



1-to Many



Many-to-1



Many-to-Many

Translation to relational model?

Translating ER Diagrams with Key Constraints

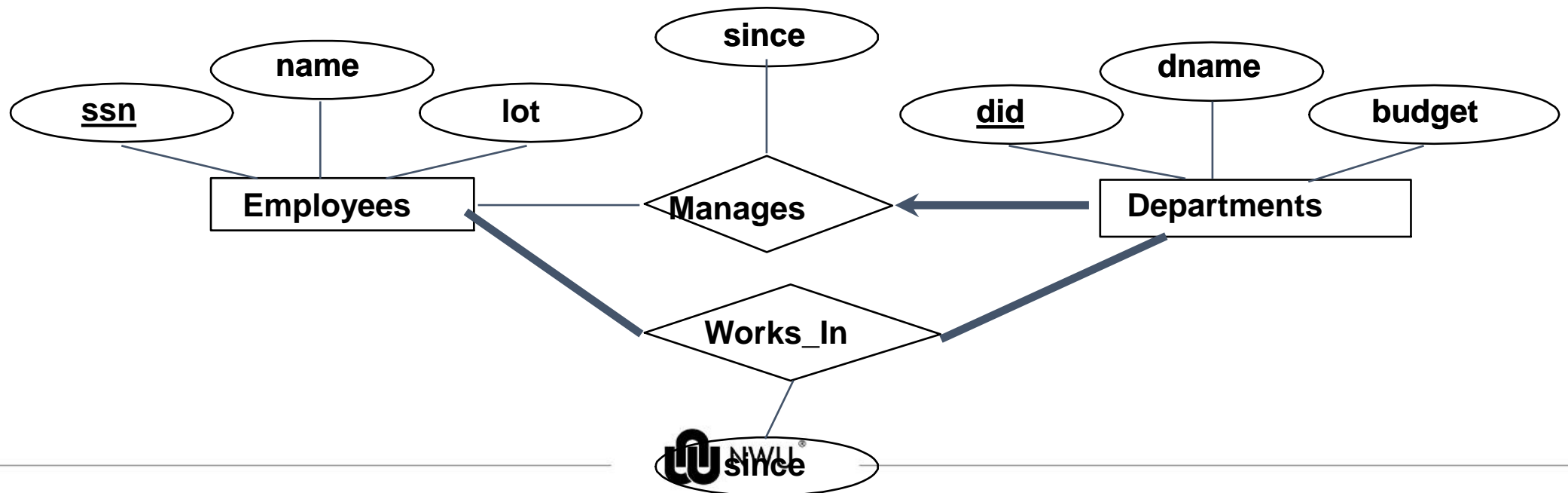
- Map relationship to a table:
 - Note that **did** is the key now!
 - Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.

```
CREATE TABLE Manages(  
    ssn CHAR(11),  
    did INTEGER,  
    since DATE,  
    PRIMARY KEY (did),  
    FOREIGN KEY (ssn) REFERENCES Employees,  
    FOREIGN KEY (did) REFERENCES Departments)
```

```
CREATE TABLE Dept_Mgr(  
    did INTEGER,  
    dname CHAR(20),  
    budget REAL,  
    ssn CHAR(11),  
    since DATE,  
    PRIMARY KEY (did),  
    FOREIGN KEY (ssn) REFERENCES Employees)
```


Review: Participation Constraints

- Does every department have a manager?
 - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total (vs. partial)*.
 - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



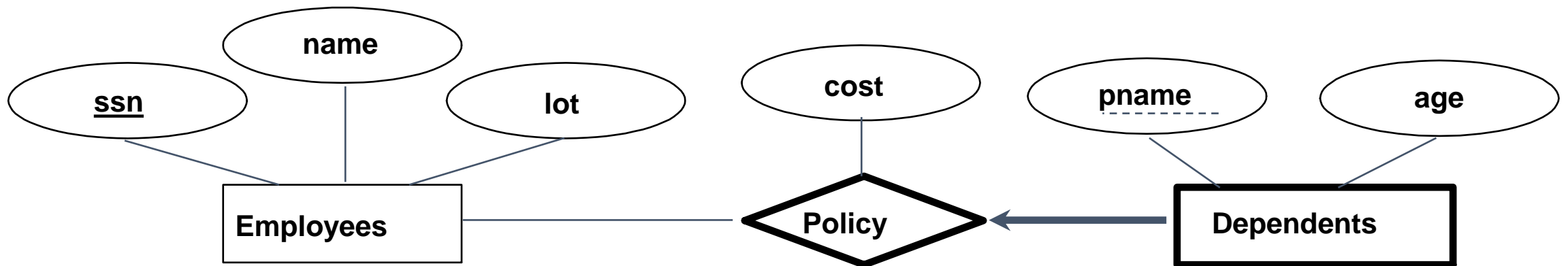
Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```
CREATE TABLE Dept_Mgr(  
    did INTEGER,  
    dname CHAR(20),  
    budget REAL,  
    ssn CHAR(11) NOT NULL,  
    since DATE,  
    PRIMARY KEY (did),  
    FOREIGN KEY (ssn) REFERENCES Employees,  
    ON DELETE NO ACTION)
```

Review: Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
 - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
 - Weak entity set must have total participation in this *identifying* relationship set.



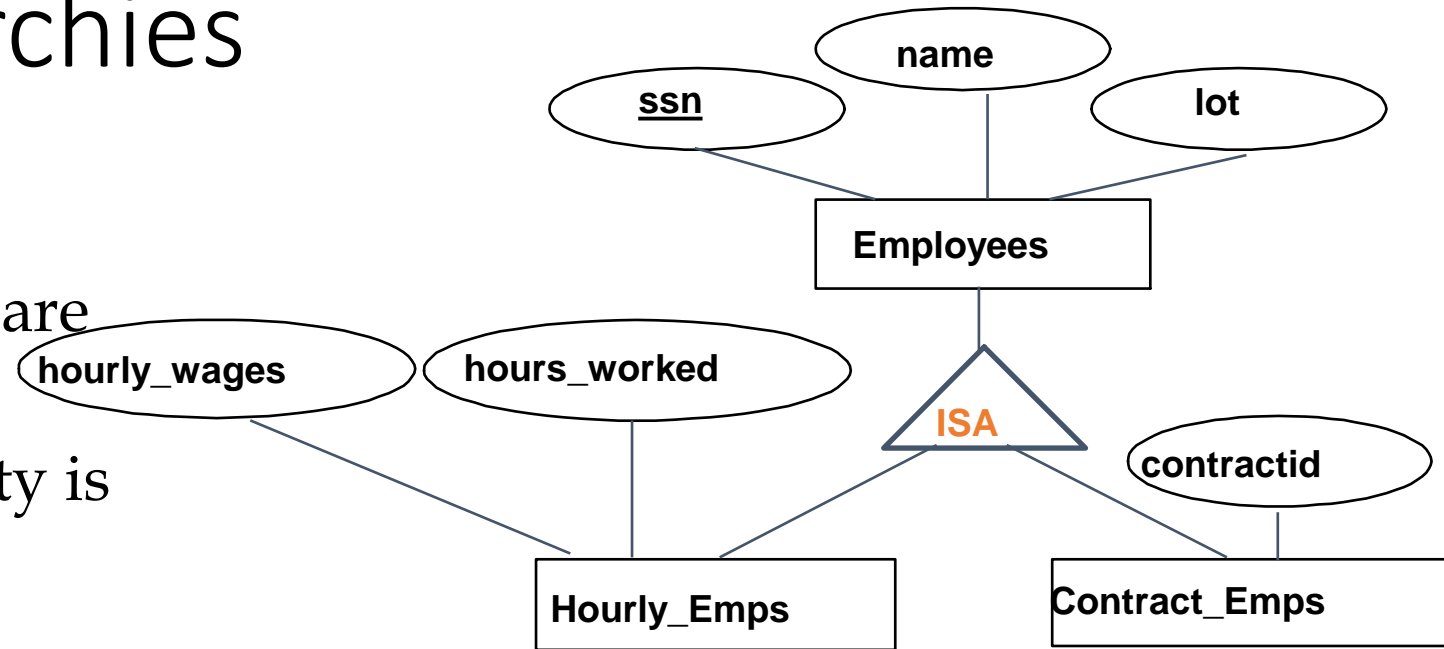
Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
 - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE Dep_Policy (  
    pname CHAR(20),  
    age INTEGER,  
    cost REAL,  
    ssn CHAR(11) NOT NULL,  
    PRIMARY KEY (pname, ssn),  
    FOREIGN KEY (ssn) REFERENCES Employees,  
    ON DELETE CASCADE)
```

Review: ISA Hierarchies

- ❖ As in C++, or other PLs, attributes are inherited.
- ❖ If we declare A **ISA** B, every A entity is also considered to be a B entity.



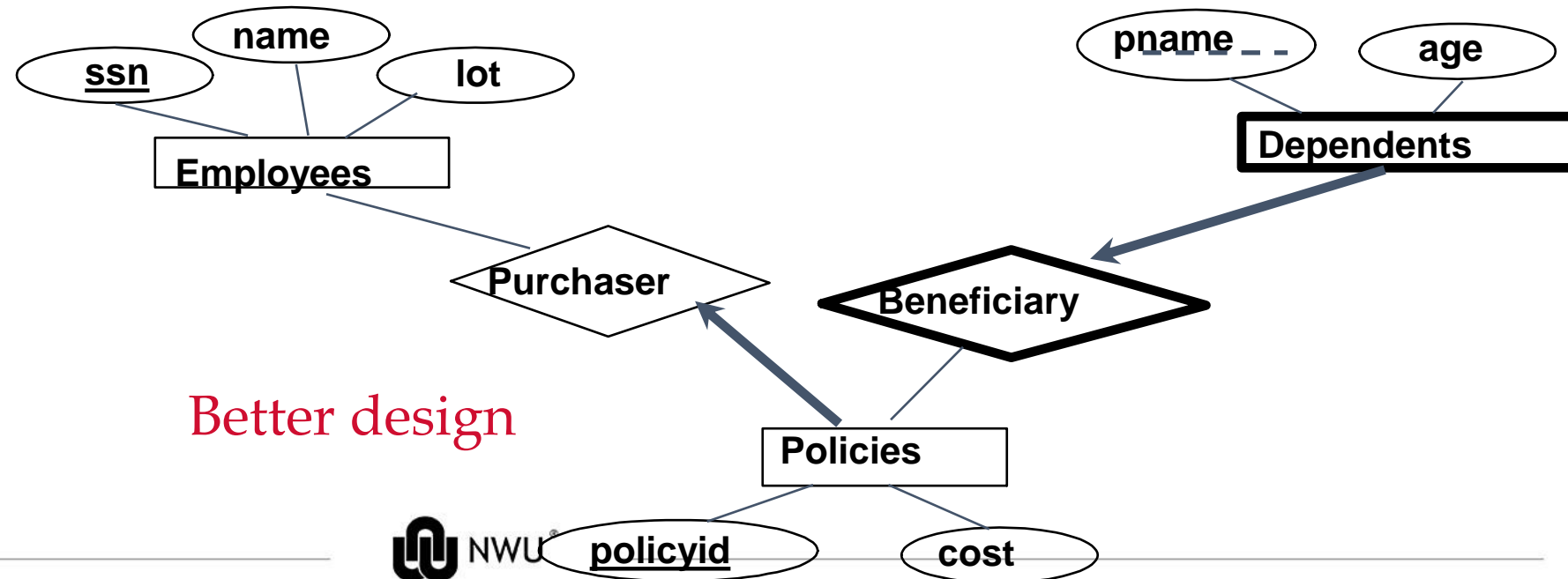
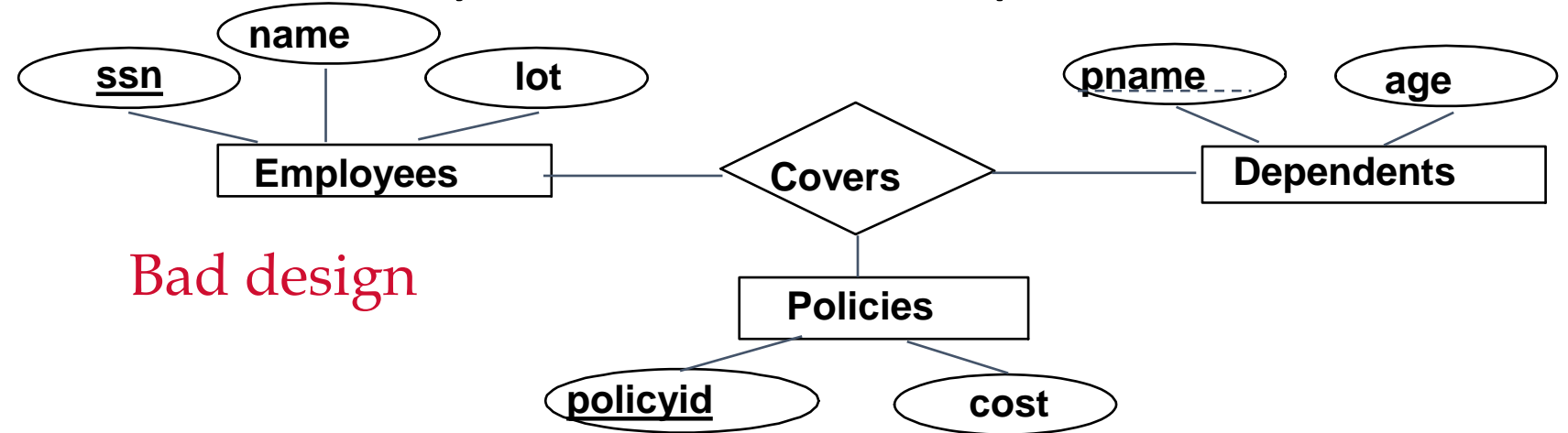
- **Overlap constraints:** Can Joe be an Hourly_Emps as well as a Contract_Emps entity? *(Allowed/disallowed)*
- **Covering constraints:** Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*

Translating ISA Hierarchies to Relations

- **General approach:**
 - 3 relations: *Employees*, *Hourly_Emps* and *Contract_Emps*.
 - *Hourly_Emps*: Every employee is recorded in *Employees*. For hourly emps, extra info recorded in *Hourly_Emps* (*hourly_wages*, *hours_worked*, *ssn*); must delete *Hourly_Emps* tuple if referenced *Employees* tuple is deleted).
 - Queries involving all employees easy, those involving just *Hourly_Emps* require a join to get some attributes.
- **Alternative: Just *Hourly_Emps* and *Contract_Emps*.**
 - *Hourly_Emps*: *ssn*, *name*, *lot*, *hourly_wages*, *hours_worked*.
 - Each employee must be in one of these two subclasses.

Review: Binary vs. Ternary Relationships

- What are the additional constraints in the 2nd diagram?



Binary vs. Ternary Relationships (Contd.)

- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to **NOT NULL** constraints.
- What if Policies is a weak entity set?

```
CREATE TABLE Policies (
```

```
  policyid INTEGER,  
  cost REAL,  
  ssn CHAR(11) NOT NULL,  
  PRIMARY KEY (policyid).  
  FOREIGN KEY (ssn) REFERENCES Employees,  
  ON DELETE CASCADE)
```

```
CREATE TABLE Dependents (
```

```
  pname CHAR(20),  
  age INTEGER,  
  policyid INTEGER,  
  PRIMARY KEY (pname, policyid).  
  FOREIGN KEY (policyid) REFERENCES Policies,  
  ON DELETE CASCADE)
```


Views

- A view is just a relation, but we store a *definition*, rather than a set of tuples.

```
CREATE VIEW YoungActiveStudents (name, grade)
AS SELECT S.name, E.grade
FROM Students S, Enrolled E
WHERE S.sid = E.sid and S.age < 21
```

- ❖ Views can be dropped using the **DROP VIEW** command.
 - How to handle **DROP TABLE** if there's a view on the table?
 - DROP TABLE command has options to let the user specify this.

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
 - Given YoungStudents, but not Students or Enrolled, we can find students *s* who have are enrolled, but not the *cid*'s of the courses they are enrolled in.

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
 - Two important ICs: primary and foreign keys
 - In addition, we *always* have domain constraints.
- Powerful and natural query languages exist.
- Rules to translate ER to relational model